Loop Optimization using Hierarchical Compilation and Kernel Decomposition

D.Barthou[1] S.Donadio[23] P.Carribault[23] A.Duchateau[1] W. Jalby[13]

[1]University of Versailles, France

[2]Bull SA Company, France

[3]CEA/DAM

CGO 2007

Outline

Library code generation for monocore architectures

- Motivation
- Description of the approach
- Kernel Decomposition
- Experiments
- Concluding remarks

Motivation

High performance linear algebra library for monocore architectures

- Automatic generation: ATLAS, PhiPAC.
  - Uses algorithmic knowledge,
  - Optimizes first for cache usage,
  - Explores optimization space by empirical search or model.
- Hand-tuned assembly: constructor library (MKL, ESSL), Goto's BLAS.

Motivation

High performance linear algebra library for monocore architectures

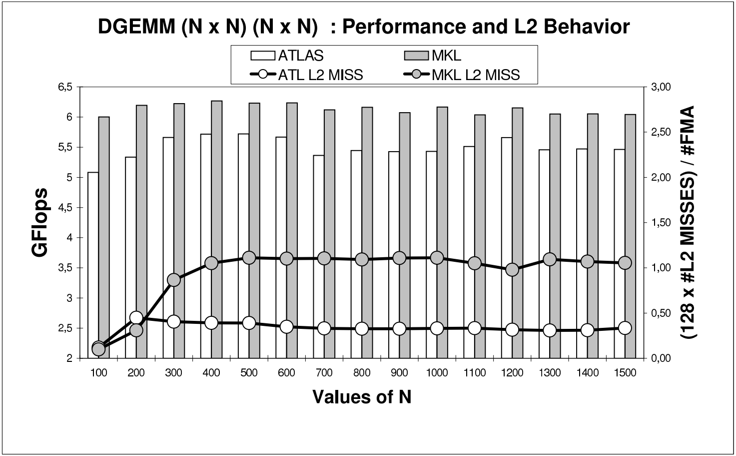
- Automatic generation: ATLAS, PhiPAC.
  - Uses algorithmic knowledge,
  - Optimizes first for cache usage,
  - Explores optimization space by empirical search or model.
- Hand-tuned assembly: constructor library (MKL, ESSL), Goto's BLAS.

Hand-tuned code outperforms ATLAS (Itanium/Pentium).
Is there something missing in compilers and/or ATLAS ?

Performance Analysis MKL/ATLAS: L2 misses

ATLAS version 5.6, MKL version 8.02 on Itanium

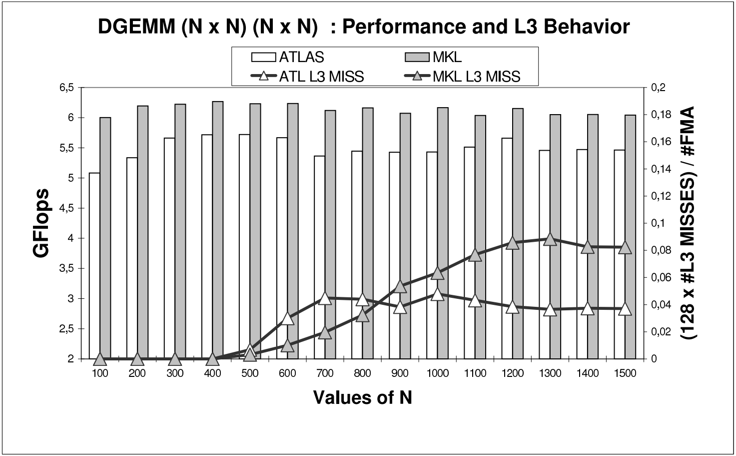
ICC compiler v9.0



Performance Analysis MKL/ATLAS: L3 misses

ATLAS version 5.6, MKL version 8.02 on Itanium

ICC compiler v9.0



Proposed Approach

Find a tradeoff between ILP and locality

1. Tile the code for data locality (if any)
2. Improve ILP of tile code
   - Apply sequences of source optimizations
   - Decompose code into simple source kernels
   - Optimize kernels with compiler and test
3. Choose the best kernel to build the best tile
   - Adapt tile size to kernel size

Kernel Decomposition

Tile for data locality

- Constraint tile sizes

Kernel Decomposition

Tile for data locality

- Constraint tile sizes

Explore optimization space on tile code

- Loop transformations
  - unroll (to improve IPC)
  - interchange (to change locality)
  - strip mine (to generate loops with constant bounds)
- Select inner loops
- Data layout transformations
  - scalar promotion (to reduce TLB misses and simplify address computation)

Kernel Decomposition

Tile for data locality

- Constraint tile sizes

Explore optimization space on tile code

- Loop transformations
  - unroll (to improve IPC)
  - interchange (to change locality)
  - strip mine (to generate loops with constant bounds)
- Select inner loops
- Data layout transformations
  - scalar promotion (to reduce TLB misses and simplify address computation)

Drive optimizations and parameters with X-language [LCPC05]

- Exhaustive search on unrolling factors, interchanges.
- Selected loop bound values

Kernel Optimization

Kernels tuned with two parameters:

- Loop bound values
  - Unrolling factor, SWP parameters, ...
- Array alignments
  - Vectorization
  - Memory bank conflicts

Rely on compiler for:

- Vectorization
- Register allocation
- Dependence analysis
- Instruction scheduling

Example of Decompositions for DGEMM

- Original tile

```
for (i = 0; i < NI; i++)
 for (j = 0; j < NJ; j++)
  for (k = 0; k < NK; k++ )
   c[i][j] += a[i][k] * b[k][j];
```

Example of Decompositions for DGEMM

- Original tile

```
for (i = 0; i < NI; i++)
 for (j = 0; j < NJ; j++)
  for (k = 0; k < NK; k++ )
   c[i][j] += a[i][k] * b[k][j];
```

- Unroll i and j loops

```
for (i = 0; i < NI; i+=2)
 for (j = 0; j < NJ; j+=2 )
  for (k = 0; k < NK; k++)
   c[i][j] += a[i][k] * b[k][j];
   c[i+1][j] += a[i+1][k] * b[k][j];
   c[i][j+1] += a[i][k] * b[k][j+1];
   c[i+1][j+1] += a[i+1][k] * b[k][j+1];
```

Example of Decompositions for DGEMM

- Original tile

```
for (i = 0; i < NI; i++)
 for (j = 0; j < NJ; j++)
  for (k = 0; k < NK; k++ )
   c[i][j] += a[i][k] * b[k][j];
```

- Unroll i and j loops

```
for (i = 0; i < NI; i+=2)
 for (j = 0; j < NJ; j+=2 )
  for (k = 0; k < NK; k++)
   c[i][j] += a[i][k] * b[k][j];
   c[i+1][j] += a[i+1][k] * b[k][j];
   c[i][j+1] += a[i][k] * b[k][j+1];
   c[i+1][j+1] += a[i+1][k] * b[k][j+1];
```

- Extracted kernel: dotproduct

```
for (k = 0; k < NK; k++)
 c00 += a0[k] * b0[k];
 c10 += a1[k] * b0[k];
 c01 += a0[k] * b1[k];
 c11 += a1[k] * b1[k];
```

dotproduct nm

```
for(i = 0 ; i < ni ; i++)
   c11 += a1[i] * b1[i];
   ...
   c1n += a1[i] * bn[i];
   c21 += a2[i] * b1[i];
   ...
   cmn += am[i] * bn[i];
```

Example of Decompositions for DGEMM

- Original tile

```
for (i = 0; i < NI; i++)
 for (j = 0; j < NJ; j++)
  for (k = 0; k < NK; k++ )
   c[i][j] += a[i][k] * b[k][j];
```

- Interchange j,k, Unroll i and k loops

```
for (i = 0; i < NI; i+=2)
 for (k = 0; k < NK; k+=2 )
  for (j = 0; j < NJ; j++)
   c[i][j] += a[i][k] * b[k][j];
   c[i+1][j] += a[i+1][k] * b[k][j];
   c[i][j] += a[i][k+1] * b[k+1][j];
   c[i+1][j] += a[i+1][k+1] * b[k+1][j];
```

dotproduct nm

```
for(i = 0 ; i < ni ; i++)
   c11 += a1[i] * b1[i];
   ...
   c1n += a1[i] * bn[i];
   c21 += a2[i] * b1[i];
   ...
   cmn += am[i] * bn[i];
```

Example of Decompositions for DGEMM

- Original tile

```
for (i = 0; i < NI; i++)
 for (j = 0; j < NJ; j++)
  for (k = 0; k < NK; k++ )
   c[i][j] += a[i][k] * b[k][j];
```

- Interchange j,k, Unroll i and k loops

```
for (i = 0; i < NI; i+=2)
 for (k = 0; k < NK; k+=2 )
  for (j = 0; j < NJ; j++)
   c[i][j] += a[i][k] * b[k][j];
   c[i+1][j] += a[i+1][k] * b[k][j];
   c[i][j] += a[i][k+1] * b[k+1][j];
   c[i+1][j] += a[i+1][k+1] * b[k+1][j];
```

- Extracted kernel: daxpy

```
for (j = 0; j < NJ; j++)
 c0 += a00 * b0[j];
 c1 += a10 * b0[j];
 c0 += a01 * b1[j];
 c1 += a11 * b1[j];
```

dotproduct nm

```
for(i = 0 ; i < ni ; i++)
   c11 += a1[i] * b1[i];
   ...
   c1n += a1[i] * bn[i];
   c21 += a2[i] * b1[i];
   ...
   cmn += am[i] * bn[i];
```

daxpy nm

```
for(i = 0 ; i < ni ; i++)
   c1[i] += a11 * b1[i];
   ...
   c1[i] += a1n * bn[i];
   c2[i] += a2n * b1[i];
   ...
   cm[i] += amn * bn[i];
```

Example of Decompositions for DGEMM

- Original tile

```
for (i = 0; i < NI; i++)
 for (j = 0; j < NJ; j++)
  for (k = 0; k < NK; k++ )
   c[i][j] += a[i][k] * b[k][j];
```

- Permute i and k

```
for (k = 0; k < NK; k++)
 for (i = 0; i < NI; i++ )
  for (j = 0; j < NJ; j++)
   c[i][j] += a[i][k] * b[k][j];
```

dotproduct nm

```
for(i = 0 ; i < ni ; i++)
   c11 += a1[i] * b1[i];
   ...
   c1n += a1[i] * bn[i];
   c21 += a2[i] * b1[i];
   ...
   cmn += am[i] * bn[i];
```

daxpy nm

```
for(i = 0 ; i < ni ; i++)
   c1[i] += a11 * b1[i];
   ...
   c1[i] += a1n * bn[i];
   c2[i] += a2n * b1[i];
   ...
   cm[i] += amn * bn[i];
```

Example of Decompositions for DGEMM

- Original tile

```
for (i = 0; i < NI; i++)
 for (j = 0; j < NJ; j++)
  for (k = 0; k < NK; k++ )
   c[i][j] += a[i][k] * b[k][j];
```

- Permute i and k

```
for (k = 0; k < NK; k++)
 for (i = 0; i < NI; i++ )
  for (j = 0; j < NJ; j++)
   c[i][j] += a[i][k] * b[k][j];
```

- Extracted kernel: outerproduct

```
for (i = 0; i < NI; i++)
 for (j = 0; j < NJ; j++)
  c[i][j] += a[i] * b[j];
```

dotproduct nm

```
for(i = 0 ; i < ni ; i++)
   c11 += a1[i] * b1[i];
   ...
   c1n += a1[i] * bn[i];
   c21 += a2[i] * b1[i];
   ...
   cmn += am[i] * bn[i];
```

daxpy nm

```
for(i = 0 ; i < ni ; i++)
   c1[i] += a11 * b1[i];
   ...
   c1[i] += a1n * bn[i];
   c2[i] += a2n * b1[i];
   ...
   cm[i] += amn * bn[i];
```

outerproduct n

```
for (i = 0; i < ni ; i++)
   for (j = 0; j < nj ; j++)
     c[i][j] += a1[i] * b1[j];
     ...
     c[i][j] += an[i] * bn[j];
```

Kernel Properties

Kernel Performance

- Independent of the application context,
- Only depends on cache level of data.

Additional benefits of kernels

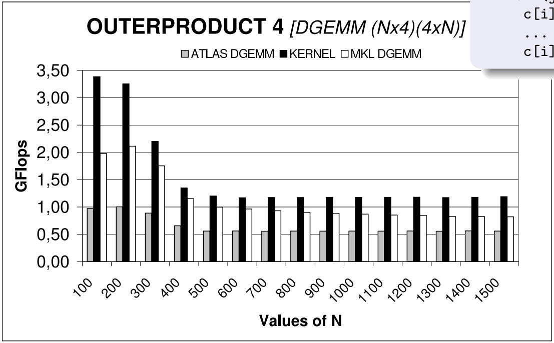
- Execution time much lower than for whole application,
- Possible reuse among different applications.

Kernel Performance on Pentium4

Pentium4 Prescott 2.8Ghz, 16KB L1, 1MB L2

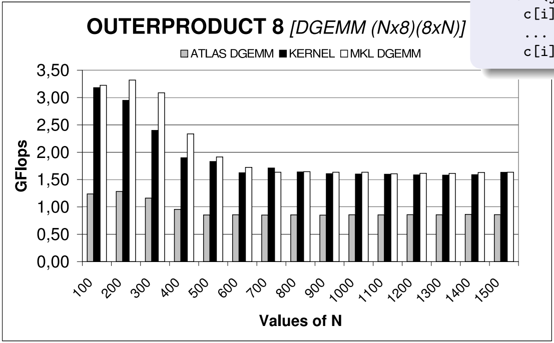
outerproduct n

```
for (i = 0; i < ni ; i++)
   for (j = 0; j < nj ; j++)
     c[i][j] += a1[i] * b1[j];
     ...
     c[i][j] += an[i] * bn[j];
```



Kernel Performance on Pentium4

Pentium4 Prescott 2.8Ghz, 16KB L1, 1MB L2

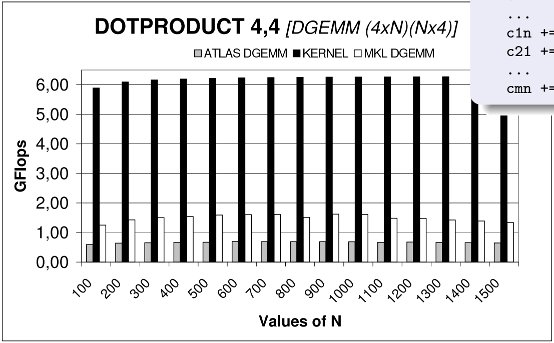


outerproduct n

```
for (i = 0; i < ni ; i++)
   for (j = 0; j < nj ; j++)
     c[i][j] += a1[i] * b1[j];
     ...
     c[i][j] += an[i] * bn[j];
```

Kernel Performance on Itanium

Itanium2 Madison 1.6GHz, 256KB L2, 9MB L3

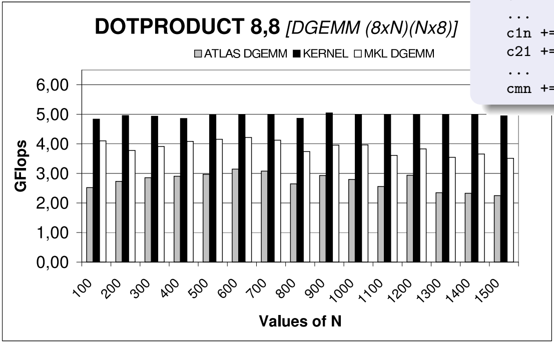


dotproduct nm

```
for(i = 0 ; i < ni ; i++)
   c11 += a1[i] * b1[i];
   ...
   c1n += a1[i] * bn[i];
   c21 += a2[i] * b1[i];
   ...
   cmn += am[i] * bn[i];
```

Kernel Performance on Itanium

Itanium2 Madison 1.6GHz, 256KB L2, 9MB L3



dotproduct nm

```
for(i = 0 ; i < ni ; i++)
    c11 += a1[i] * b1[i];
    ...
    c1n += a1[i] * bn[i];
    c21 += a2[i] * b1[i];
    ...
    cmn += am[i] * bn[i];
```

Kernel Composition

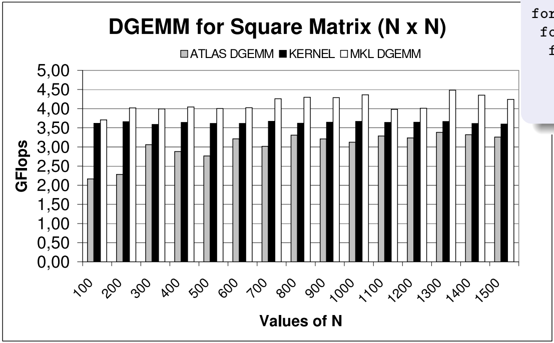
Build the best performing tile:

- For each possible kernel, add copies/transpositions (if necessary)
- Select best kernel (with copy times)
- Choose a tile size multiple of kernel size

Predict global performance out of:

- kernel measured performance,
- memory copies/transpositions measured performance

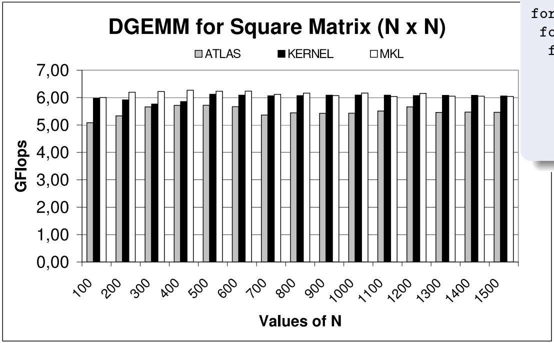
Performance Results for Pentium4



dgemm

```
for(I = 0; I < NI; I+=ni)
 for(J = 0; J < NJ; J+=nj)
  for(K = 0; K < NK; K+=nk)
   for(i = 0 ; i < ni; i++)
    for(k = 0 ; k < nk ; k++)
      daxpy44(&c[i],a[i][k],&b[k]
```

Performance Results for Itanium



dgemm

```
for(I = 0; I < NI; I+=ni)
 for(J = 0; J < NJ; J+=nj)
  for(K = 0; K < NK; K+=nk)
   // copy a,c and transpose b
   for(i = 0 ; i < ni ; i++)
    for(j = 0 ; j < nj; j++)
     dotprod44(&c[i][j],&a[i],&b[j])
   // copy-out c
```

Summary

Proposed Approach

- Not application dependent,
- Code generation
  - No assembly code,
  - Only classical optimizations and compiler technology,
  - Very competitive with MKL, outperforming ATLAS,
  - Works for rectangular matrices,
- Exploration space
  - Optimization parameters: (unrolling factors, interchange, selection of inner loops, loop bound values, alignment)
  - Execution of all kernels
  - No execution for whole code (within 1% of predicted time)

Future Works

How to further guide the search?

- Avoid execution
  - Filtering assembly codes
  - Matching previously executed kernels (reuse)
- Make exploration space smaller
  - Model performance

Extend to multicore codes

Thank You !

L2 and L3 performance impact